Ryan Barr
Jung Bok Cho
Yangyang Liu
Sen Yang

CPSC 5042 Milestone 1: Outline for a 'Hangman' style game

Our project is a hangman style game where the user guesses letters until the word is revealed or they run out of attempts. Points are awarded for the number of attempts left when the word is correctly identified. For example: if the word was "cat", the round started with 3 points, and the user guessed the letter "r" they would lose a point and be prompted to guess again. If they then guessed "a" they would be shown "_ a _" and would not lose a point. If they then guessed the whole word "cat" they would be awarded 2 points for the round. Points will be based on the length of the word being guessed.

Ideally this will be a 2-player game, where 1 player submits the word to be guessed, player 2 guesses, and after player 2 uses all their guesses or correctly guesses the word, the roles reverse and player 1 guesses the word player 2 submits. After a set number of rounds the point totals will determine a winner of the 2 players.

RPCs:

loginRPC – requests user credentials and creates a connection to the server (implemented)

ARGNAME	DataType	Input/Output	Example
sock	Int	Input	13
status	Int	Output	1 for success, else -1
error	String	Output	Detailed error information (optional)

disconnectRPC – disconnects client from the server, and closes client sockets (implemented)

ARGNAME	DataType	Input/Output	Example
sock	Int	Input	13
status	Int	Output	1 for successful disconnect, else -1
error	String	Output	Detailed error information (optional)

guessLetter – player guesses a letter in the secret word

ARGNAME	DataType	Input/Output	Example
letter	char	Input	'r'
secretWord	string	Output	"c _ r _ _ r" if the user had guessed c and r previously
points	int	Output	Points remaining

guessWord – player guesses the secret word

ARGNAME	DataType	Input/Output	Example
word	string	Input	"career”
status	int	Output	1 if guess was correct, else -1
points	int	Output	Points remaining

score – gets the players current score, or both players if playing 2 player

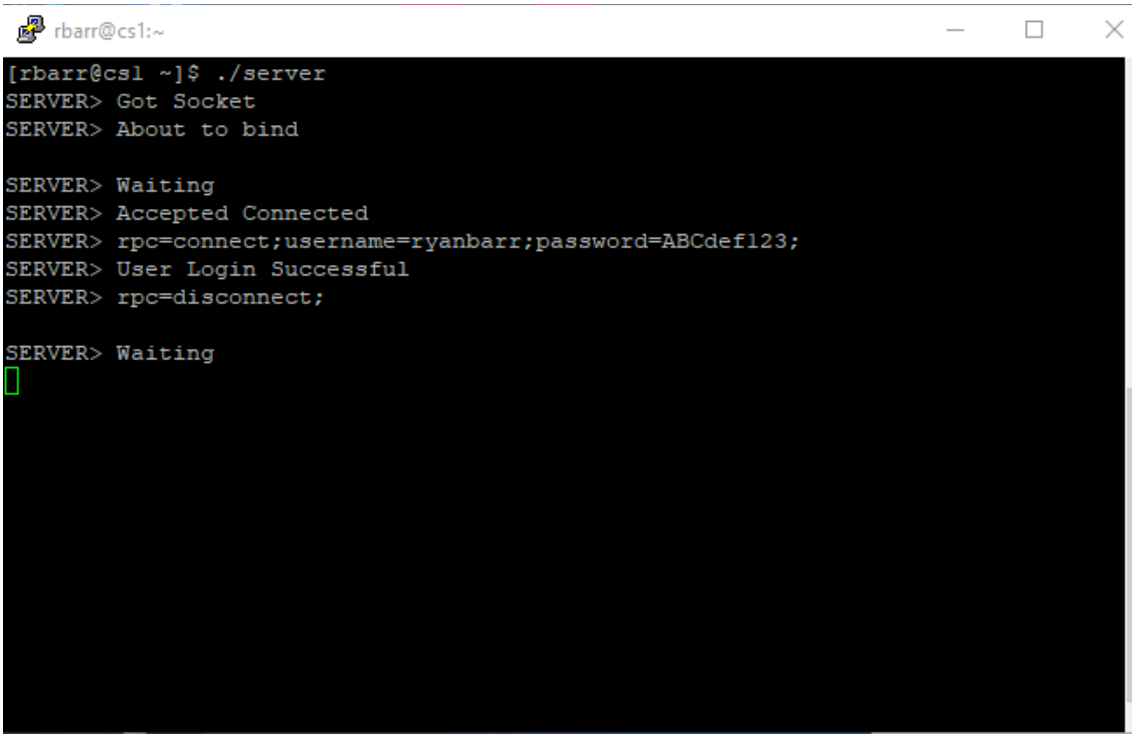
ARGNAME	DataType	Input/Output	Example
player1Score	int	Output	12
player2Score	int	Output	16

end – ends the current game, displaying the score and which player won (if 2 player)

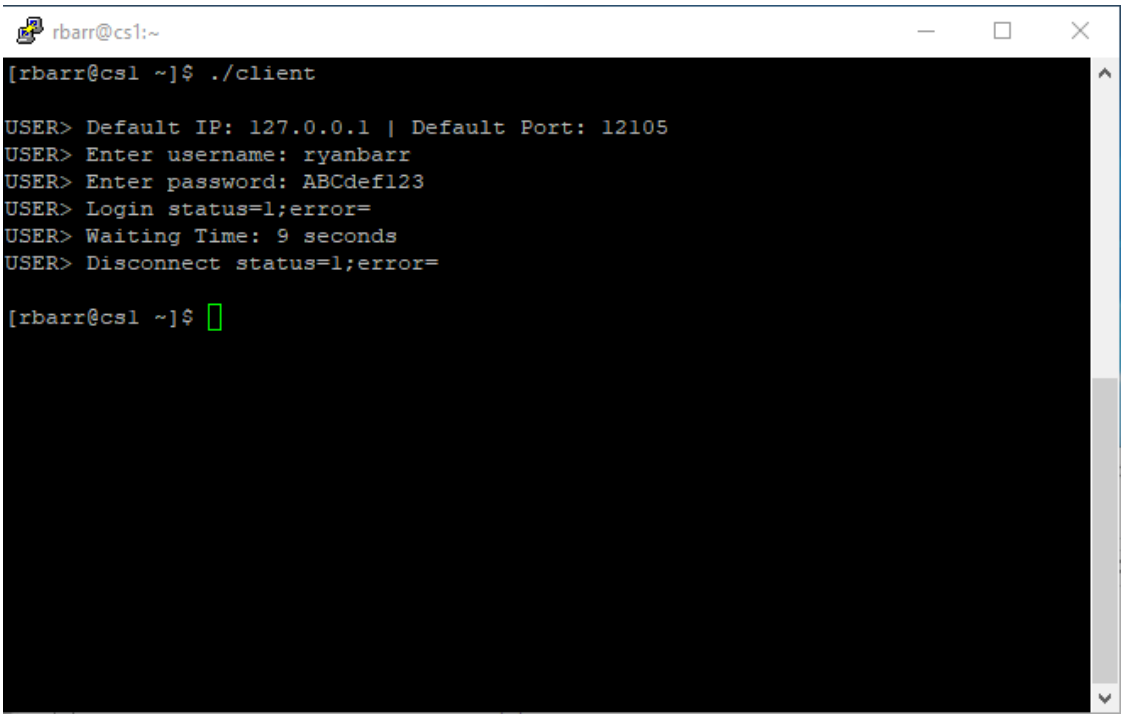
ARGNAME	DataType	Input/Output	Example
player1Score	int	Output	12
player2Score	int	Output	16
winner	string	Output	"Cho”

Screen Shots:

Server



Client



Optional modifications to .cpp files:

We have included several versions of our server and client that operate slightly differently under the hood. The server.cpp and client.cpp files are documented with appropriate comments to use, but here is a summary:

1) Basic mode:
Compiled as-is the server and client will communicate with basic string RPCs which are divided up into a vector using the `splitBuffer` function in server.cpp

2) JSON mode:
If you delete the comment markers on line 26 of server.cpp and line 22 of client.cpp, to enable the `#define JSONFORMAT` tags, the server and client will communicate by passing JSON packets. It will use the `jsonToVector` function as well as `authenticate`, `initializeUsers` and `login` functions .

3) Object Oriented mode:
If you delete the comment markers on line 29 of server.cpp to enable the `#define OOFORMAT` tag, the server and client will communicate using the `KeyValue` and `RawKeyValueString` Classes to pass RPC strings.

Makefile, Compilation, and execution instructions:

On CS1, run the following commands to compile and run the attached makefile, server.cpp and client.cpp files:

Put makefile, server.cpp and client.cpp in the same dir, then run the following commands:

```
make
make clean
```

Alternately you can manually compile using the following:

```
g++ server.cpp -o server
g++ client.cpp -o client
```

After compiling using either method, simply execute the following:

```
./server
./client
```

It is strongly recommended that client and server be run in separate console windows for clarity of function

User names are as follows:

BASIC MODE AND OBJECT ORIENTED MODE:

Username: Any 6+ char's, all lower case. Example: "**administrator**"
Password: Any 8+ char's, must use uppercase, lowercase and number. Example "**Password1"**

JSON MODE:

Users are hardcoded into a map for this mode. See `initializeUsers()` for full list.
Username: "Sen"
Password: "12345Abc"